マルチメディア通信と分散処理　52－35
（1991．9．27）

弱同期によるフォールトトレラントプロセス

桧垣博章　　曽根岡昭直

NTT ソフトウェア研究所

概要

同期実行する複製バックアッププロセスを用い、故障発生時にマスタの切替えを行なうことによりフォールトトレランスが実現される。しかし、分散環境ではメッセージ伝達遅延のために、複製との完全な同期は困難である。本稿では、複製プロセスを以下のように弱く同期させることによるフォールトトレラント化方式を提案する。(1) メッセージ受信の順序を同一にして、バックアップの実行履歴をマスタと同じにする。(2) バックアップの送信イベントの実行をマスタより先行させない制御によって、故障発生時のメッセージ送信の重複や欠落を防ぐ。本方式は、チェックポイント設定や切替え時のロールバック、複製プロセス以外への故障通知が不要である。

Weakly synchronized fault-tolerant processes

Hiroaki Higaki　　Terunao Soneoka

NTT Software Laboratories
3-9-11 Midori-cho, Musashino-shi, Tokyo 180, Japan

Abstract

Fault-tolerance can be achieved by synchronizing replicated processes and letting a backup process take over the failed master process. However, it is difficult to synchronize them closely in distributed systems. This paper proposes a new method for achieving fault-tolerance by *weakly synchronizing* processes: (1)Let every replica have the same event history by letting every message be received in the same order by every replica. (2)Let every send event be executed exactly once in the replicated processes by prohibiting backup processes from being ahead of the master process at every send event.

This method needs not take checkpoints, roll back, and broadcast every failure to every process.

1 はじめに

プロセスの冗長構成によって、故障発生時でも処理を継続できるフォールトトレランスの実現法は、大きく次の2方式に分類することができる [8]。

ソフトウェア方式 マスタプロセスがバックアッププロセスにチェックポイントデータを転送する。故障発生時にバックアッププロセスはチェックポイントへロールバックして実行を開始する。

ハードウェア方式 複製プロセスを密に同期させて、処理が同時実行されるようにする。故障発生時には、ただちにバックアップに切り替える。

ソフトウェア方式では、チェックポイントの設定記述に要するソフト開発負担が大きいという欠点を持つことが言われている。また、ロールバックを行なうために処理の中断時間が長い。この点、ハードウェア方式は同期実行しているので中断時間は短い。しかし、密結合のマルチプロセッサにおいてではなく、プロセスが互いに通信して処理を進める分散環境で、プロセスの冗長化によるフォールトトレランスを実現する場合、メッセージの伝達遅延のためにハードウェア方式のような強い同期を厳密に実現するのは困難である。本稿では、分散環境におけるハードウェア方式フォールトトレランスのソフト的な実現法を検討する。

分散環境でのフォールトトレランス実現法として、仮想同期を用いる方式がある [2] 。これは、複製プロセスからなるプロセスグループをつくり、グループへの送信メッセージを構成プロセスが同一順序で受信するように制御することで、仮想的な同期を実現するものである。しかし、この方式では構成プロセスの故障通知と他グループプロセスからのメッセージ受信との順序への考慮はしているが、故障通知と他グループプロセスへのメッセージ送信との順序への対処がない。そのため、バックアップへの切り替え時にメッセージ送信の重複や欠落といった矛盾を生じることがあり、チェックポイント－ロールバックによる障害回復が必要になっている。

そこで本稿では、メッセージ送信とプロセス故障を適切に順序化する制御を実現する弱同期方式を提案する。本方式は、送信イベントでバックアップがマスタに対して先行しない、マスタ交替を引継送信イベントまで遅延させる、という弱い同期によって、プロセス故障によるマスタ切り替え時に生じるメッセージ送信の矛盾を防ぐ。また、

- チェックポイント設定を必要とせず、プロセスの処理はロールバックしない。
- プロセス故障通知をグループに隠蔽する。

といった特徴により、コストの低下を実現している。

2 システムモデル

プロセスは分散環境にあり、相互作用は非同期型のメッセージの通信によってなされる。メッセージ伝達遅延は時間分布しているが、タイムアウトによる故障検出は可能であると仮定する。また下位レイヤの機能によりメッセージの紛失、複製、書換は起こらないものと仮定する。

プロセスには、状態遷移機械モデルを仮定する。プロセスが実行するイベントは、他のプロセスからメッセージを受信する受信イベント、メッセージを送信する送信イベント、それ以外のローカルイベントのいずれかである。プロセスは、他のプロセスから送信されたメッセージを受信イベントの実行により受け取る。そして、現在の状態と受信メッセージのみによって定まるイベント列を実行して状態遷移を行ない、再び受信イベントを実行する。イベント列はいくつかのローカルイベントと送信イベントからなる。このイベント列を実行している最中に他のプロセスから送信されたメッセージはバッファリングされる。後の受信イベントでは、このバッファからメッセージが受け取られる。

プロセスは複数のメッセージ送信をアトミックに実行することができる。このとき、これらのメッセージが全て宛先プロセスに送信されるか、この送信プロセスの故障によって全く送信されないかのいずれかであり、一部のメッセージだけが送信されることはない。

また、因果関係を保ったメッセージ通信が可能である。イベント間の因果関係 ($\rightarrow$) とは次の条件を満足するものである [5]。

1. 同一プロセスでイベント e と f がこの順に実行されるとき、$e \rightarrow f$ である。
2. メッセージ m について、$send(m) \rightarrow receive(m)$ である。
3. $e \rightarrow f$ かつ $f \rightarrow g$ ならば $e \rightarrow g$ である。

また因果関係を保ったメッセージ通信とは、以下の条件を満足するものである [3]。

> メッセージ m, n について $send(m) \rightarrow send(n)$ が成り立つとき、両メッセージを受信するプロセスにおいて、$receive(m) \rightarrow receive(n)$ が成り立つ。

プロセスの故障は、'fail-stop' のみを仮定する。これは、以下の条件をみたすものである。

- 故障プロセスは停止する。故障後に誤ったメッセージを送信して他のプロセスに影響を及ぼすことはない。
- 故障は他のプロセスによって検知される。*

システムの耐故障性を高めるために、各プロセスを N 重に複製したプロセスグループをつくる。グループに含まれるプロセスの Process-ID (PID) の集合をグループのビューという。グループに対してメッセージを送信する場合には、このビューを参照してそれぞれのプロセスへのメッセージ送信を行なう。

3 仮想同期

3.1 仮想同期とは

非同期分散環境では、グループに含まれる複製プロセスに同じメッセージが送信されても、メッセージの伝達遅延やプロセスの実行速度が異なるために受信時刻がプロセスごとに異なり、複製を常時同一状態に保つことは困難である。

そこで、同一グループのプロセスではメッセージの受信順序を同一にすることで、同一の状態遷移履歴を持つように制御する方法が提案されており、これらのプロセスは仮想同期しているという。これを実現している代表的なシステムである ISIS[4] では、メッセージ受信順序を同一にする ABCAST という2フェーズのメッセージ通信を用いている。第1フェーズで送信プロセスから受信グループの全プロセスにメッセージが送信される。このとき受信メッセージは Undeliverable 状態でバッファリングされ、受信イベントによって受け取ることはできない。第2フェーズで受

*サイトが故障していない場合のプロセス故障は、例えば OS のモニタにより検知される。また、サイト間で "I'm alive" メッセージを定期的に送受信することで、タイムアウトを用いたサイト故障の検知が可能である [3]。故障通知メッセージはこれらの故障検出機構から送信されるが、故障後も他プロセスの読み出しが可能な Stable-strage[6] に Logical-clock の値を記録しておくことで因果関係を保つことが可能である。

信順序を同一にするためのメッセージが送信され、これを受信してDeliverable状態になったメッセージを定められた順番に受け取ることができる。

また、プロセス故障が起こった場合には、このプロセスが含まれていたグループのビューを変更する必要があり、故障通知はシステムの全てのプロセスに対して送信される。この故障通知メッセージも他のメッセージと同様、グループ構成プロセスが同一順序で受信するように制御される。さらに、故障プロセスが送信したメッセージ以降にこの故障通知が受け取られるように制御することによって、矛盾の発生を防いでいる。ISIS システムにおいては、GBCAST というメッセージ通信によってこれを行なっている。

3.2 仮想同期の問題点

仮想同期では、メッセージ送信の実行については何も制御を行なっていないため、各複製プロセスがメッセージ送信を行ない、次のような問題を生じる。

1. 冗長化していないプロセス間の1つのメッセージ転送に対応する N 重化グループ間の転送メッセージ数が、$O(N^2)$ である。また、同一イベントでそれぞれの複製プロセスから送信されたメッセージも受信順序を同一にするため、余分なコストを要する。
2. 複製プロセスから送信された同一メッセージを受信イベントで受け取らないようにするために、メッセージ受信についての履歴を保存しなければならず、このための記憶領域が必要である。

そこで、複製プロセスの1つをマスタプロセスとし、それ以外をバックアッププロセスとする。送信イベントではマスタだけが実際にメッセージの送信を行ない、バックアップはメッセージ送信をせずに後続イベントを実行する。また、マスタが故障した場合は、故障通知を受け取ったバックアップのうちの1つを新しいマスタとする。これによって故障のない場合にはメッセージの重複送信を避けることができ、メッセージ数は $O(N)$ になる。しかし、仮想同期による実現では故障発生時に次のような問題点がある。

1. マスタプロセスの故障によって、メッセージ送信の実行が重複あるいは欠落することがある (図 1)。プロセス故障が通知

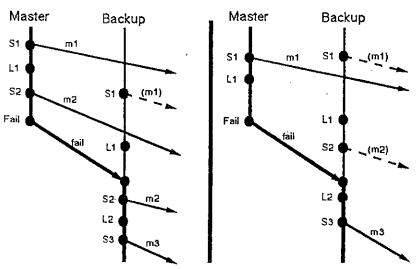

図 1: 送信の重複と欠落

されたときに、マスタがどのメッセージまで受け取ってどの送信イベントまで実行したのを、バックアッププロセスは知ることができない。したがって、新たにマスタとなったプロセスがどの送信イベントからメッセージ送信を行なうべきか確定できない。

2. マスタになったプロセスがどの送信イベントからメッセージ送信を開始すべきか知ることができたとしても、故障通知時点でのこのプロセスのイベント実行状況は、故障したマスタプロセスとは同一とは限らない。もし、この送信イベントまで実行が進んでいない場合には、メッセージ送信の重複を避けるためにマスタになるのを延期する制御が必要である。また、バックアップとしてメッセージ送信をせずにこの送信イベントを既に終了している場合、このプロセスがこのまま実行を継続するとメッセージ送信の欠落が起こる。これらを避けるために、バックアッププロセスが全送信イベントの履歴を取る方法が考えられるが、十分な記憶領域の確保が必要である。また、チェックポイントを設定してロールバックさせる方法もあるが [1] 、これらの矛盾を解消するチェックポイントの設定法、チェックポイントデータ保存のための記憶領域の確保とプロセスロールバックのオーバーヘッド、といった問題点がある。
3. 1つのプロセスが故障するたびにシステムの全プロセスに通知するのはコストが大きい。プロセス故障が通知されていないために送信プロセスが送信に用いるビューに故障プロセスが含まれているとしても、そのメッセージが破棄されるだけであり、正常プロセスへのメッセージ送信には影響しない。また、グループからの送信メッセージについては、受信プロセスでは送信したグループだけが問題であり、送信したプロセスを知る必要はないためプロセス故障の通知の必要性はない。プロセス故障によるビュー更新の影響への適切な対処機構によって、プロセス故障をシステムの全プロセスに通知する必要がなくなり、故障対処に要するコストを削減することができる。

4 弱同期フォールトトレラントプロセス

次の性質を持つ複製プロセスからなるプロセスグループによって、前章で述べた問題点が解決できる。これを弱同期フォールトトレラントプロセスとよぶ。

1. グループ間メッセージの送信は、各送信イベントにつき1度だけ実行される。プロセス故障によって送信メッセージの欠落や重複は起こらない。
2. グループの構成プロセスはグループ間通信メッセージを同一順序で受け取る。これによって構成プロセスが同一履歴の状態遷移を行なうことになる。
3. プロセス故障時に、正常プロセスがロールバックすることはない。したがって、メッセージ履歴やチェックポイント設定といった過去の状態の保存を行なう必要がない。
4. それぞれのプロセス故障はグループに隠蔽される。

5 メッセージ制御法

前章で提案した弱同期フォールトトレラントプロセスを実現するメッセージ制御法について述べる。

5.1 送信非先行制御と交替遅延制御

プロセス故障によるマスタの交替を行なう場合でも、メッセージ送信が欠落や重複することなく実行されるためには、以下のよ

うな制御を行なえばよい。

1. 全ての送信イベントについてバックアップがマスタに対して先行しない。
2. 以前のマスタが行なった最後の送信イベントまでの送信は行なわない。

前者は、送信イベント実行時に、マスタがメッセージ送信とともに送信済みをバックアップに通知し、バックアップがこれを受信してメッセージ送信がなされたことを知るまで後続イベントを実行しないことで達成される。後者は、グループの構成プロセスの故障通知メッセージとこの送信済みメッセージの受信の因果関係を保存し、マスタ交替を送信イベントとの順序関係を満たすように遅延させることでなされる (図 2)。このとき、いずれのプロセスも決してロールバックすることがなく、チェックポイントの設定などを必要としない。また、マスタの実行は決してブロックされず、バックアップがマスタより先行実行している場合に送信イベントでブロックされるが、それまでの受信イベントとローカルイベントの実行が束縛されることはない。

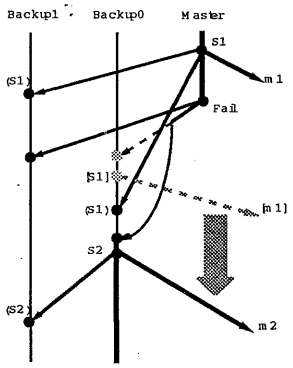

図 2: 送信重複の解消

5.2 同一順序受信

グループ間のメッセージ受信は、マスタの定めた順序に構成プロセスが従うことで同一順序化する (図 3)。マスタはメッセージを受信した順に Deliverable としてバッファリングし、この順に受信イベントで受け取る。バックアップは他グループからの受信メッセージを Undeliverable で一旦バッファリングし、マスタからの順序決定のメッセージを受信してはじめて Deliverable とする。故障のない限り 1 つのマスタが定めた順に Deliverable になるので、同一順序が実現するのは明らかである。一方、故障が発生したときには故障通知と順序決定のメッセージが因果関係を保存していれば、故障したマスタが定めた順序は全ての構成プロセスに伝えられており、順序を定められていないメッセージは Undeliverable でバッファリングされているので、新しいマスタに順序決定される。こうして、全ての受信メッセージが 1 つのプロセスによって 1 回だけ順序を決定され、全ての構成プロセスに伝えられる。

回復プロセスがグループに追加されているが、それが送信プロセスの持つビューに反映されていないとき、グループに送信されたメッセージはこのプロセスには送信されない。そこで、送信に用いられたビューに含まれていないプロセスへはマスタによってフォワードし、受信プロセスではメッセージ受信と順序決定を同時に行なうこととする。このとき、同時に送信プロセスに対して変更されたビューを送り返し、以降のメッセージ送信にはこれを用いさせる。

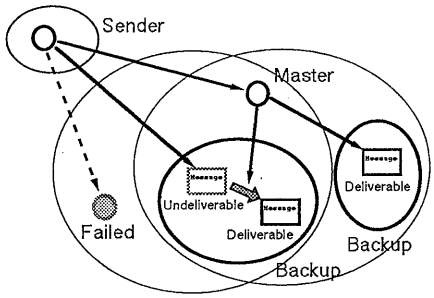

図 3: グループ間通信

5.3 故障通知

各機構実現のためにはメッセージの送信に用いるビューの変更が必要であり、故障プロセスを含むグループの全ての構成プロセスに対してはプロセス故障を通知しなければならない。このときにグループ内通信の他のメッセージと因果関係を保つ必要があることは、これまでに述べた通りである。

他グループのビューの更新は前節で述べたように、メッセージ受信時に更新されたビューを返信する方法によって実現する。受信から次のビュー受信までの間に故障 / 回復したグループ構成プロセス数が k_e 以下であり、構成プロセスのうちの同時故障数が k_i 以下のとき、$N > k_e + ki$ ならばグループ間メッセージの送受が確実に行われることが保証できる (証明は後述)。この方法ではそれぞれのプロセス故障 / 回復が全プロセスに通知されるわけではないので、メッセージ数を減少することができる。

6 アルゴリズム

6.1 準備

弱同期フォールトトレラントプロセス実現のために以下の準備を行なう。

1. 全てのプロセスが次のものを持つ (図 4) 。(以降の説明では括弧内の略称を用いる。)

(a) メッセージを保持する無限長の 2 つのキューと 1 つのバッファ。

- Send-message-queue (S-queue)
- Receive-message-queue (R-queue)
- Pending-message-buffer (P-buffer)

(b) グループ構成プロセスの PID からなる 2 種類のビュー。

- 自グループのビューである Group-view (G-view)。構成プロセスの順位と状態 (Alive/Dead) の情報が含まれている。

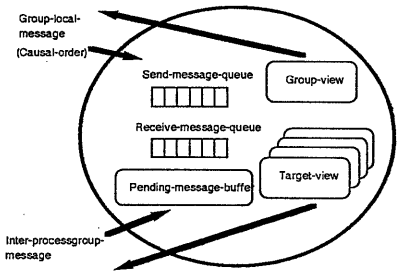

図 4: 構成図

全構成プロセスが、全て Alive 状態である同一ビューを持つように初期設定する。
- 他グループのビューである Target-view(T-view)。

2. グループ間メッセージには Message-ID(MID) を付与する。
3. グループ内メッセージ通信は因果関係を保存する。†

6.2 受信マスタと送信マスタ

送信イベントでグループ間通信メッセージを送信する送信マスタと、メッセージの受信順序を決める受信マスタは次のようにして決定する。

- G-view で最上位の Alive 状態である複製プロセスが受信マスタ (R-master) として動作する。
- G-view で最上位の複製プロセスが送信マスタ (S-master) として動作する。

6.3 送受信イベント

'受信イベント'と'送信イベント'のアルゴリズムは次の通りである。

受信イベント

1. R-queue にメッセージがない、または先頭メッセージが Undeliverable である間はブロックされる。
2. R-queue の先頭にある Deliverable メッセージをデキューして受信メッセージとして受け取り、終了する。

送信イベント

- S-master である場合、以下のメッセージ送信をアトミックに実行して終了する。
  1. 送信先グループの T-view に含まれるプロセスに対して、この T-view を付与したメッセージを送信する。
  2. G-view に含まれている Backup-process に対して、送信済み通知メッセージを送信する。
- Backup-process である場合。
  1. S-queue にメッセージが含まれていない間はブロックされる。
  2. S-queue の先頭メッセージをデキューしていずれかを行なう。

(a) このメッセージが送信済み通知ならば、メッセージ送信を行なわずに終了する。

(b) このメッセージがプロセスの故障通知ならば、G-view からこの故障プロセスを削除してはじめに戻る。変更によって S-master となることがある。

6.4 グループ間通信

他グループプロセスから送信されたメッセージを、受信グループの構成プロセスが同一順序で受け取るためのアルゴリズムを示す。

他グループプロセスからのメッセージの受信

- R-master である場合。
  1. 受信メッセージを Deliverable 状態で R-queue の最後尾にエンキューする。
  2. 以下のメッセージ送信をアトミックに実行する。

(a) 受信メッセージに付与されている T-view に含まれている Backup-process に対して、受信メッセージと同じ MID を付与した Ordering メッセージを送信する。

(b) G-view に含まれているが、受信メッセージに付与されている T-view に含まれていない Backup-process が存在するならば、このプロセスに対して Forward メッセージを送信するとともに、送信グループに対して G-view を含んでいる view 更新メッセージを送信する。

- Backup-process である場合、R-queue に同一 MID を持つ Undeliverable である Ordering メッセージが
  1. 含まれていない場合、この受信メッセージを P-buffer に加える。
  2. 含まれている場合、この Ordering メッセージを Deliverable にする。受信メッセージは破棄する。

メッセージ同一順序受信

- Backup-process が R-master からのメッセージを受信したとき、いずれかを行なう。
  1. Ordering メッセージであるならば、R-queue の最後尾にエンキューする。このとき、このメッセージと同じ MID を持つ受信メッセージが P-buffer に

(a) 含まれているならば、この Ordering メッセージを Deliverable 状態にして、P-buffer に含まれていた受信メッセージを破棄する。

(b) 含まれていないならば、この Ordering メッセージを Undeliverable にする。

  2. Forward メッセージであるならば、R-queue の最後尾にエンキューし Deliverable にする。

他グループからの view 更新メッセージの受信

- T-view をメッセージに含まれる G-view に変更する。

6.5 プロセス故障

グループ構成プロセスの故障通知の受信アルゴリズムを示す。

† N 重化プロセスグループに対して、N 次元のベクトルクロック [7] を用いて実現できる。

故障通知の受信

1. 受信したプロセス故障通知メッセージを S-queue の最後尾にエンキューする。
2. G-view の故障プロセスの状態を Dead にする。もし、この変更によって R-master になったときには、P-buffer に含まれる全てのメッセージについて以下を実行する。

(a) R-queue の最後尾にエンキューし Deliverable にする。
(b) 以下のメッセージ送信をアトミックに実行する。

i. メッセージに付与されている T-view に含まれる Backup-process に対して、このメッセージと同じ MID を付与した Ordering メッセージを送信する。
ii. G-view に含まれていて、メッセージに付与されている T-view に含まれていない Backup-process が存在するならば、このプロセスに対して Forward メッセージを送信し、送信グループに対して G-view を含む view 更新メッセージを送信する。

6.6 プロセス回復

プロセスの回復は、故障プロセスとは異なる PID を持った新しいプロセスの生成により実現される。回復プロセスは PID を付与した Recover メッセージをグループに対して送信する。受信したプロセスでは、これを他グループからの受信メッセージとほぼ同様に扱うが、以下の点が異なる。

- R-master が Recover メッセージを受信したときには、回復プロセスを G-view の最下位に Alive 状態で追加する。さらに、Ordering メッセージと Forward メッセージの送信とともに、状態情報 (状態変数 ,R-queue , S-queue ,P-buffer ,G-view ,T-view) を回復プロセスへアトミックに送信する。回復プロセスはこれを受信したときに、R-queue に含まれている Undeliverable 状態のメッセージを全て Deliverable にする。
- Backup-process は R-master からの Ordering メッセージまたは Forward メッセージを受信したときに、回復プロセスを G-view の最下位に Alive 状態で追加する。
- Deliverable になった Recover メッセージは破棄する。
- R-master になったときに P-buffer に Recover メッセージが含まれているならば、回復プロセスを G-view の最下位に Alive 状態で追加して、状態情報と Ordering/Forward メッセージの送信を行なう。

7 正当性の証明

前節で示したアルゴリズムが、弱同期フォールトトレラントプロセスの 4 つの性質のうち、

3. プロセスのイベント実行がロールバックされていない。
4. 他グループプロセスに対して、全てのプロセス故障通知をしていない。

の 2 つを満たしていることは明らかである。ここでは、

1. 送信が 1 回だけなされる。
2. メッセージ受信順序が同一である。

という残りの 2 つの性質が満たされていることを示す。

7.1 送信制御

送信イベントでメッセージ送信が 1 度だけ実行されることを証明する。

送信イベントでメッセージ送信の欠落がない

$N > k_i$ であるから全てのグループ構成プロセスが故障することはない。

プロセスが送信イベントを終了するのは、メッセージ送信を行なった場合か、他のプロセスから送信済み通知を受信した場合のいずれかである。したがってメッセージ送信がなされなければ、いずれのグループ構成プロセスも後続のイベントを実行することはない。

また、全ての構成プロセスがバックアップとなり、送信済み通知の受信待ちのまま停止し続けることもない。このようなことがあるならば、グループ内チャネルにも構成プロセスの S-queue にも故障通知メッセージがなく、どの正常プロセスも G-view の最上位にないことになる。しかし、全ての故障通知メッセージが受信されていることから G-view には Dead のプロセスはない。また、G-view は同一順序に初期化されており、回復プロセスは同一順序で追加されるので、構成プロセスの持つ G-view は同一である。したがって、G-view の最上位が自プロセスである、つまり S-master であるプロセスが存在することになり、これは仮定に矛盾する。

以上により、送信イベントではグループからのメッセージ送信が必ず行なわれる。

送信イベントでメッセージ送信の重複がない

異なるバックアップの故障通知メッセージの間には因果関係がないため、G-view の更新履歴はグループ構成プロセスで必ずしも同一ではない。ところが、バックアップが送信マスタとなるのは、G-view の全ての上位プロセスから故障通知メッセージを受信し、S-queue の先頭からデキューしたときであるから、複数の構成プロセスが送信マスタになることは起こらない。

また故障によってマスタが交替するとき、メッセージ送信が重複実行されるのは送信済み通知より故障通知の方が先に S-queue にエンキューされる場合であるが、グループ内通信では因果関係が保存されることからこのようなことは起こらない。

したがって、メッセージは重複送信されない。

7.2 同一順序メッセージ受信

全構成プロセスに受信順序が通知される

メッセージ m の送信に用いられるビューを T-view(m)、m の Ordering/forward メッセージ (以下順序化メッセージと略す) の送信に用いるビューを G-view(m) とする。

G-view(m) に含まれるプロセスは正常であるか、故障しているが故障通知がマスタに届いていないかのいずれかである。また回復プロセスは、マスタが状態情報送信以降に受信したメッセージについて受信する必要があり、その時点では必ず G-view に含まれている。したがって、G-view(m) には順序化メッセージを受信する必要がある全てのプロセスが含まれている。

一方、T-view(m) は前回のビュー更新メッセージで得たものであり、同時故障数が k_i 以下であるから $|\text{T-view(m)}| > N - k_i > k_e$。また、T-view(m) の送信から G-view(m) の送信までに故障 / 回復によって交替したグループ構成プロセス数が k_e 以

下であることから、|T-view(m)∩G-view(m)| > 0 が成り立つ。さらに、G-view(m)−T-view(m) には T-view(m) の送信以降に回復したプロセスだけが含まれており、これらは T-view(m) に含まれる全てのプロセスよりも下位である。したがって、受信マスタは T-view(m) ∩ G-view(m) に含まれている。以上により、m の順序化メッセージの送信は必ず行なわれる。

受信順序が重複決定されない

同一マスタによって重複決定されないのは明らかである。あるプロセスが受信マスタとなるのは G-view の全ての上位プロセスが Dead になったとき、すなわち全ての上位プロセスから故障通知を受信したときである。グループ内プロセス間通信メッセージは因果関係を保持することから、マスタによって送信された順序化メッセージは、全ての構成プロセスでこのマスタの故障通知より必ず先に受け取られる。したがって、マスタが交替しても以前のマスタが決定した順序情報は新しいマスタに必ず受け取られているので、重複決定されることはない。

受信順序は構成プロセスで同一

同一マスタから送信された順序化メッセージが同一順序で受信されることは明らかである。故障によるマスタ交替が起こる場合にも、故障前のマスタによる順序化メッセージ送信と故障後の新しいマスタによる順序化メッセージ送信とは故障通知を介した因果関係がある (図 5)。したがって、順序メッセージの受信順序は

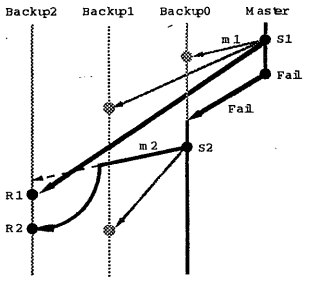

図 5: 故障時の同一順序受信

故障のある場合も同一である。

8 評価

グループ構成プロセス数 N、システム構成プロセス数 M(一般に $M \gg N$) のときに必要なメッセージ数を評価する。ISIS の仮想同期方式の場合、グループ間のメッセージ通信 (ABCAST) では $3N$ であり、故障通知では M プロセスに送信されるうえ順序制御も行なうので、総メッセージ数は $3M$ となる [3]。これに対して弱同期方式のメッセージ通信では、送信メッセージに N、送信済み通知に $N-1$、受信順序決定に $N-1$ であるから、合計 $3N-2$ である。また故障通知については、故障検知時に必要であるのはグループ構成プロセスへの通知のみで $N-1$ である。この他に他グループプロセスへビュー更新通知が必要であり、ビュー変更通知間に故障 / 回復する平均プロセス数を k' とすると 1 回の故障あたり平均 $(M-N)/k'$ である。

また、プロセス間メッセージの伝達遅延を Δ としたときの通信による遅延は、ISIS の仮想同期ではグループ間メッセージ、故障通知とも 3Δ である [3] のに対して、弱同期方式の場合は、メッセージ通信では 2Δ、故障通知では Δ である。

9 まとめ

本稿では、分散環境におけるプロセスの複製によるフォールトトレランスを実現する方法として、弱同期方式を提案し、その実現アルゴリズムの提示と正当性の証明を行なった。

本方式はメッセージの同一順序受信、メッセージ送信におけるバックアップの非先行制御といった性質を持つ。これによって、プロセス故障時の送信メッセージの重複や欠落といった矛盾の発生が解消された。また、チェックポイント設定の必要がなく、故障通知はグループに隠蔽される、といった低コスト化の特徴を持つ。

今後の検討課題としては

- チャネルでのメッセージ紛失等があるモデルにおける実現法 (下位レイヤを含めた検討)。
- 同一 PID でプロセス回復が可能であるモデルにおける故障を完全に隠蔽した実現法。

が考えられる。

参考文献

[1] K.Birman, T.Joseph, T.Raeuchle and A.Abbadi, "Implementing Fault-Tolerant Distributed Objects," IEEE Trans. on Software Engineering, Vol.SE-11, No.6, pp.502-508 (1985).

[2] K.Birman and T.Joseph, "Exploiting Virtual Synchrony in Distributed Systems," Proc. of the 11th ACM Symposium on Operating System Principles, pp.123-138 (1987).

[3] K.Birman and T.Joseph, "Reliable communications in the presence of failures," ACM Trans. on Computer Systems, Vol.5, No.1, pp.47-76 (1987).

[4] K.Birman and K.Marzullo, "ISIS and Meta Project," Sun Technology, Vol.2, No.1, pp.90-104 (1989).

[5] L.Lamport, "Time, clocks, and the ordering of events in adistributed system," Commun. of the ACM, Vol. 21, No. 7, pp.558-565 (1978).

[6] F.Schneider, "Byzantine Generals in Action: Implementing Fail-Stop Processors," ACN Trans. on Computer Systems, Vol.2, No.2, pp.145-154 (1984).

[7] A.Schiper, J.Eggli, and A.Sandoz, "A new algorithm to implement causal ordering," Lecture Notes in Computer Science, Vol.392, *Distributed Algorithms*, pp.219-232 (1989).

[8] 渡辺, " フォールトトレラントデザインの概要," bit, Vol.19, No.9, pp.1201-1217 (1987).